CSE 451: Section 2 Handout
1/16/2020

File Descriptors and Intro to Multi-Processing

File Descriptors:

Why do we want processes to have their own sets of file descriptors?

Draw out the process open file table layout after the following c code:

```
	int fd1 = open("file.txt", O_RDONLY);
	int fd2 = open("file.txt", O_RDWR);
```

Linux supports more functions built on top of dup, `dup2` is an example.

```
int dup2(int oldfd, int newfd);
```

The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd. If the file descriptor newfd was previously open, it is silently closed before being reused. The steps of closing and reusing the file descriptor newfd are performed atomically. This is important, because trying to implement equivalent functionality using close(2) and dup() would be subject to race conditions, whereby newfd might be reused between the two steps. Such reuse could happen because the main program is interrupted by a signal handler that allocates a file descriptor, or because a parallel thread allocates a file descriptor.

Note the following points:
* If oldfd is not a valid file descriptor, then the call fails, and newfd is not closed.
* If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd.

Discuss how you would implement it in osv.

CSE 451: Section 2 Handout
1/16/2020

File Descriptors and Intro to Multi-Processing

Multi-Processing:

Take a look at `proc_spawn` in proc.c, what does it do? If a process is forked, what steps in `proc_spawn` should remain the same, and what should change?

Where will a process resume execution after coming back to user mode? (and where is this information stored?)

To differentiate the new processes from the old process, we call the new process a child of the parent old process. The return value of `fork` is different between the child and the parent. The parent will return the child process id and the child will return 0.
How can we alter the return value of the fork function to simulate the situation above? (Hint: take a look at arch/x86_64/include/arch/trap.h)